Runtime analysis

a)
```
void f1(int n)
{
    int i=2; // O(1)
    while (i<n) {
        ...; //O(1)
        i = i*i; //O(1)
    }
}
```
loop dependent on n.

example iterations: For n=100 i=2, i=4, i=16, i=256 — growth rate $\geq 2^i$ per step. So sequence is

$\boxed{\Theta(\log n)}$

b)
```
void f2(int n)
{
    for(int i=1; i<n; i++){          // n times
        if (i % (int)sqrt(n) == 0){  // Dependent on this if.
            for(int k=0; k<pow(i,3); k++) {   // takes O(n^3) 0,1,...,i^3
                ...; //O(1)
            }
        }
    }
}
```
// from outermost for loop, innerloop // only triggers $n^{1/2}$ times of n iterations

The innermost loop has a T.C. of $O(n^3)$ as the worst case has $\sum_{k=0}^{i^3} O(1)$ where i is N in the worst case. This only triggers when the if statement triggers, and it can trigger at most $\sqrt{n}$ times

so we get recall $n^{1/2} = \sqrt{n}$

$O(n) + n^{1/2} \cdot \sum_{k=0}^{i^3} O(1)$ but for $i \to n$ we get

$O(n) + n^{1/2} \cdot O(n^3) = O(n^{7/2})$ $\boxed{\Theta(n^{7/2})}$

c)
```
for (int i=1; i≤n; i++) {                          n times
  for (int k=1; k≤n; k++){                         n times
    if (A[k]==i) {                                 // Can trigger all times so ignored.
      for (int m=1; m≤n; m= m+m) {                 log n
        ...; // O(1)
      }
    }
  }
}
```

Inner most loop increases similarly to $\log_2 n$ as $(m = m+m) = (m = 2 \cdot m)$
The if statement can be triggered all times so it can be practically ignored.
each of outer 2 for loops run n to n times, so inner loop runs $n \cdot n$ times $= n^2$ and inner most loop runs $n^2 \cdot \log n$ times

So we have $\sum_{i=1}^{n} \left(\sum_{k=1}^{n} \left(O(1) + \Theta(\log(n)) \right) \right)$

($O(1)$: if statement)

$$= \Theta(n^2 \log(n))$$

d)
```
int f(int n)
{
  int *a = new int[10];
  int size = 10;
  for (int i = 0; i < n; i++)
  {
    if (i == size)
    {
      int newsize = 3*size/2;
      int *b = new int[newsize];
      for (int j = 0; j < size; j++) b[j] = a[j];
      a = b;
      size = newsize;
```

```
}
a[i] = i* i*;
  }
}
```

Most of this code seems to be misleading as the first for loop is the only one relating on n. Size is const, new size is a ratio of a const, j loops to a const, and size resizes by a factor of a const, so even with the many multiples of consts of n, being used we drop the coefficients to get

$$O(n) + O(n) + \ldots$$

however many times, but it's const w.r.t its line of code so we obtain

$$\boxed{\Theta(n)}$$